

# CSCI 53700 – Fall 2019

## Assignment Number 1

Due Date: October 8, 2019

---

This assignment is based on the principles of clock consistency, associated drifts, and logical clocks in a distributed system. You have to create a simulation, running on a single machine, of a simple distributed anomaly detection system. Such a system will consist of four Detection Nodes (DNs). Each DN can carry out three types of events – *internal*, *send*, or *receive*. There are two kinds of internal events – message generation and anomaly detection. During the message generation event, a DN will generate a random text message (exact size and its contents are not relevant for the assignment); while during the anomaly detection event, a DN will classify an incoming message as *normal* or *abnormal*. Again, the exact technique used to carry out the classification is not relevant for the assignment. It is expected that each DN will have an input and output queue to store incoming/outgoing messages. Based on some heuristics (could be even random), a DN will decide which kind of event to carry out. Each DN will contain a *logical clock*, a concept first proposed by Lamport.<sup>1</sup> Each DN will associate its event with the value of its logical clock. The concept of the logical clocks and associated synchronization, based on the following rules, will attempt to resolve the clock consistency in this system:

1. Each event (internal, send, or receive) in the system is associated with a time-stamp, based on logical clock (described below) of the corresponding DN.
2. Each DN,  $D_i$ , will have an associated logical clock,  $LC_i$ . This logical clock is implemented as a simple counter that is incremented whenever an event takes place within that DN. All the logical clocks in the system are initialized to zero. Since a logical clock has a non-negative integer value, it assigns a unique number to all events occurring in  $D_i$ . The time stamp of an event is the value of the logical clock when that event was created. Hence, if an event A occurs before an event B in  $D_i$ , then  $LC_i(A) < LC_i(B)$ .
3. As indicated above,  $D_i$  will heuristically decide if it wants to carry out an internal event or send a message to  $D_j$  or receive a message from  $D_k$ , if there is such a message present in its incoming queue.
4. If  $D_i$  decides to carry out an internal event then it can either generate a message to be sent out to  $D_j$  (see next point) or classify a received message into the normal or abnormal category. Every internal event will be associated with the current value of  $D_i$ 's logical clock.
5. If a  $D_i$  decides to send a message to a randomly selected  $D_j$  then it will attach the current value of its logical clock to that message and place in its outgoing queue.
6.  $D_i$  may also decide to receive a message sent by  $D_j$  if  $D_j$  has sent a message to  $D_i$  earlier and is in the incoming queue of  $D_i$ .

---

<sup>1</sup><http://research.microsoft.com/en-us/um/people/lamport/pubs/time-clocks.pdf>

7. Whenever  $D_j$  receives a message from  $D_i$ , it will advance its logical clock if the time stamp associated with the incoming message is greater than or equal to the current value of its logical clock, i.e., if  $D_j$ , receives a message (event B) from  $D_i$  with a time stamp  $t$  and  $LC_j(B) \leq t$  then  $D_j$  should advance its logical clock such that  $LC_j(B)$  is equal to  $t + 1$ .
8. Any DN in the system may exhibit Byzantine or arbitrary behavior at any time (perhaps, randomly chosen). For example,  $D_i$  may choose not to advance its logical clock at all when any event occurs in it.

Your task is to:

1. Propose the interaction and failure models for this system. Discuss the pros and cons of your design.
2. Simulate this entire environment in Java using threads. Allow the simulation to reach a *steady-state*, i.e., run the program for a large number of iterations. During each iteration and at the end of the simulation compare the values of the logical clocks of DNs – this comparison should indicate the clock drifts and their synchronizations. Repeat the simulation with different probabilities and access their effects on the clock drifts.
3. Create a brief report that indicates the aforementioned models and the analyses of the results of your simulation.

Please employ good software engineering principles in your design and implementation. The program must run on any machine from the following list:

```
in-csci-rrpc01.cs.iupui.edu 10.234.136.55
in-csci-rrpc02.cs.iupui.edu 10.234.136.56
in-csci-rrpc03.cs.iupui.edu 10.234.136.57
in-csci-rrpc04.cs.iupui.edu 10.234.136.58
in-csci-rrpc05.cs.iupui.edu 10.234.136.59
in-csci-rrpc06.cs.iupui.edu 10.234.136.60
```

Provide adequate documentation of your program. Create a *makefile* and a *readme* file for your program. All these files (source files, *readme*, *makefile*, and *report*) should be submitted via the *submittd* command on *tesla.cs.iupui.edu* in a zipped folder with the following format (*LastNAmeA1.zip*) - e.g., *RajeA1.zip*. Also, hand-in a hard-copy of the report at the beginning of the class on the due date.